# Systems beyond POSIX

Curtis Millar

UNSW Sydney & CSIRO's Data61

# Questions

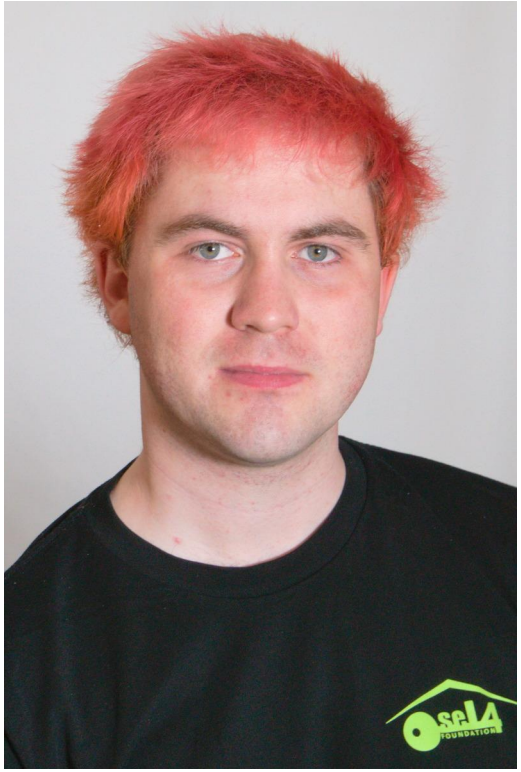
To submit questions, we are asking that you place them into the discussion board on the virtual hub page. You can also use this area to continue the discussion after the presentation as speakers will be checking for new comments throughout the summit.

# Who am I?

Curtis Millar

Research engineer in Trustworthy Systems Group at CSIRO's Data61

Mostly working on the seL4 kernel directly



# Building systems software on seL4

- Need to be able to construct and manage *systems*
- Consistent interface for common resources
  - File systems
  - Networking
  - Memory management
  - Scheduling
  - Communication
- How has this been done everywhere else?

# Portable Operating Systems Interface

- POSIX
- Application programming interface standard
- Standardisation of the interface for UNIX and its derivatives
- Implemented in many operating systems
  - System-V, Xenix, BSD, AIX, SunOS, Solaris, Linux, Darwin (macOS/iOS)...
- Deals with common OS concerns
  - Process management
  - Inter-process communication
  - Filesystems
  - Networking
  - Resource management
- Many applications already target it

# Should we provide a POSIX interface?

- We could use all of the existing POSIX application software
- POSIX (and C) are well-known
- POSIX is tried and tested

- POSIX has a number of fundamental flaws
- Extensions to POSIX have provided improved alternatives
- Some issues are too fundamental

# Why not POSIX?

- POSIX is a large collection of interfaces
- A small subset is most frequently used
  - Most examples in the wild
  - Easiest to understand interfaces
- This commonly used subset is unsuitable
  - Patterns that ensure poor performance and large overheads
  - Difficult to migrate to more performant alternatives
  - Resource management with insufficient access control
- *Typical* POSIX software is doesn't optimise and is difficult to secure

# The problem of POSIX

The primitives of POSIX software are detrimental to systems software construction and we need a more suitable alternative.

# Byte-copying I/O and IPC

# Streams and files

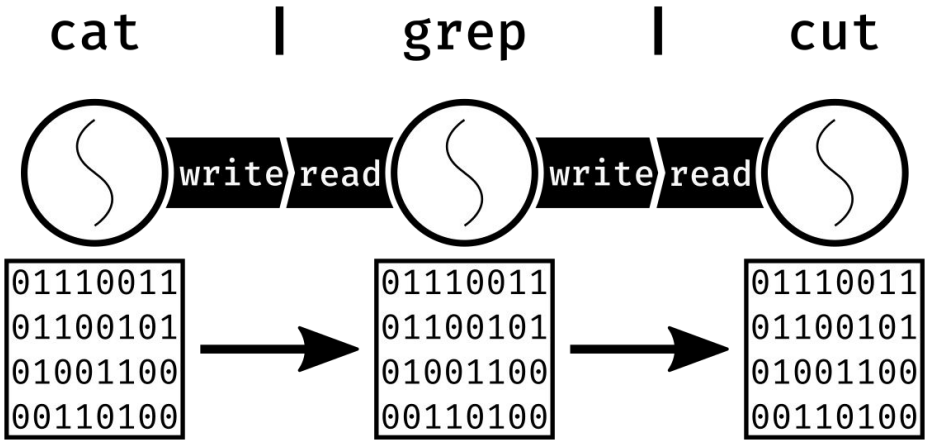
- Operate on a file descriptor
- Specify a region of bytes
- Bytes copied out on a write
- Bytes copied in on a read

```
ssize_t write(int fd, const void *buf, size_t count);
/* Also fwrite, send, sendmsg, putmsg, and mq_send */

ssize_t read(int fd, void *buf, size_t count);
/* Also fread, recv, recvmsg, getmsg, and mq_receive */
```

# Copying is expensive

- Duplication of data in memory
- Waste of CPU cycles to copy data
- Need to bring both copies of data into cache
- Cost scales with number of copies
- Particularly bad with pipelines



# Shared Memory + semaphores

- Open a shared memory object
- Map into memory
- All holders observe changes to memory
- Requires additional synchronisation
- Need to negotiate identifiers for memory object and synchronisation object

```
int shm_open(const char *name, int oflag, mode_t mode);
/* Also shmget */

void *mmap(void *addr, size_t length, int prot,
           int flags, int fd, off_t offset);

sem_t *sem_open(const char *name, int oflag,
                mode_t mode, unsigned int value);
int sem_post(sem_t *sem);
int sem_wait(sem_t *sem);
/* Also sem_trywait and sem_timedwait */
```

# Blocking interfaces

# Blocking interfaces introduce overhead

- Communication interfaces block by default
- Current thread suspends until the result is available
- Easiest way to multitask is to split tasks on different threads
  - Each task is allocated a thread
  - Task's thread blocks when it waits for input
  - Forced to deal with preemptive concurrency interleaving
- Non-blocking file descriptors or use aio calls
  - Functions return immediately
  - Wait in bulk on outstanding requests
  - Scales poorly to large numbers of pending requests
  - Often doesn't work for actual files (usually largest latencies on access)

# Global namespaces

# The default for resource sharing

- Reserve a global name
- Pass that name to other processes
  - Command line arguments
  - Environment variables
  - Via IPC
- Lookup names in namespace

# What global namespaces exist?

- The filesystem
- RT Semaphores*
- RT Message queues*
- System-V
  - XSI message queues*
  - XSI semaphores*
  - XSI shared memory segments*
- IP network
- Hostnames
- Process & thread IDs

*May just use the filesystem

# Problems of global namespaces

- Everyone can see everything
- Need to apply access control to every object
  - Unnecessarily finely grained
- Access control policy is determined by the namespace
  - Read/write/execute for user/group/other in POSIX filesystem
  - Too limited in most cases
- Not all namespaces have access control
- Global namespaces introduce covert channels
  - Anyone can add objects
- Hard to configure access controls correctly and consistently

# A better OS interface for seL4

# How to ensure a better API

- Optimal interfaces
  - Fast
  - Lightweight
  - Easy to secure
- Make the most optimal choice the default
- Make the most optimal choice as easy to use as possible
- Reduce foot-guns (or at least make them obviously unappealing)

# Minimise copy-base communication

- Transfer data in shared memory as the default
- Make it easy to construct and propagate shared-memory channels
- Provide easy-to-use synchronisation primitives
- Minimise mapping overheads
  - Reuse existing mappings as much as possible

# Semi-synchronous interfaces

- There should be *one* fundamental blocking interface
- All other interfaces should complete as much as possible synchronously and return immediately
- Use the blocking interface to wait for any outstanding request to complete
- Ensure this can scale to 10,000-100,000 requests with minimal overhead
- Compose multitasking on top of this interface
  - Use as a synchronous interface by blocking after every request
  - Provide reusable multitasking library
  - Use language-level multitasking (Go, C#, Rust, Javascript, Python…)
- Use threads for multiprocessing (avoid N:N)

# Resource management

- Share resources via IPC
- Services can apply any kind of access control policy on resources they produce
  - Implement ACLs flexibly
  - Don’t need to apply ACLs too finely or too coarsely
- Remove global namespaces provided to all processes
  - Single global filesystem
  - Global network interfaces

# Could we implement POSIX on top?

- Yes! However...
- We'd re-introduce every issue we are trying to avoid in POSIX
- Only really worthwhile to support existing POSIX applications
- Should not aim to encourage the use of POSIX or make it the primary interface
- Implement a 'POSIX service' that provides the POSIX interface to a group of processes (similar to a 'container')

# The operating system interface

- These are only the primitive needed to construct an OS
- Next need to standardise resource management & access control
- Standardise interfaces for common concerns
  - Filesystem
  - Networking
  - Human interfaces
  - etc.
- Need to ensure introspectability across this interface
  - Probably need some all-knowing root component
- Area for future experimentation and discovery!

# Questions

To submit questions, we are asking that you place them into the discussion board on the virtual hub page. You can also use this area to continue the discussion after the presentation as speakers will be checking for new comments throughout the summit.

# Further reading

- *A fork() in the road* - Andrew Baumann Jonathan Appavoo Orran Krieger Timothy Roscoe. HOTOS 2019
- *What UNIX Cost Us* - Benno Rice. LCA 2020

# Thanks for watching!